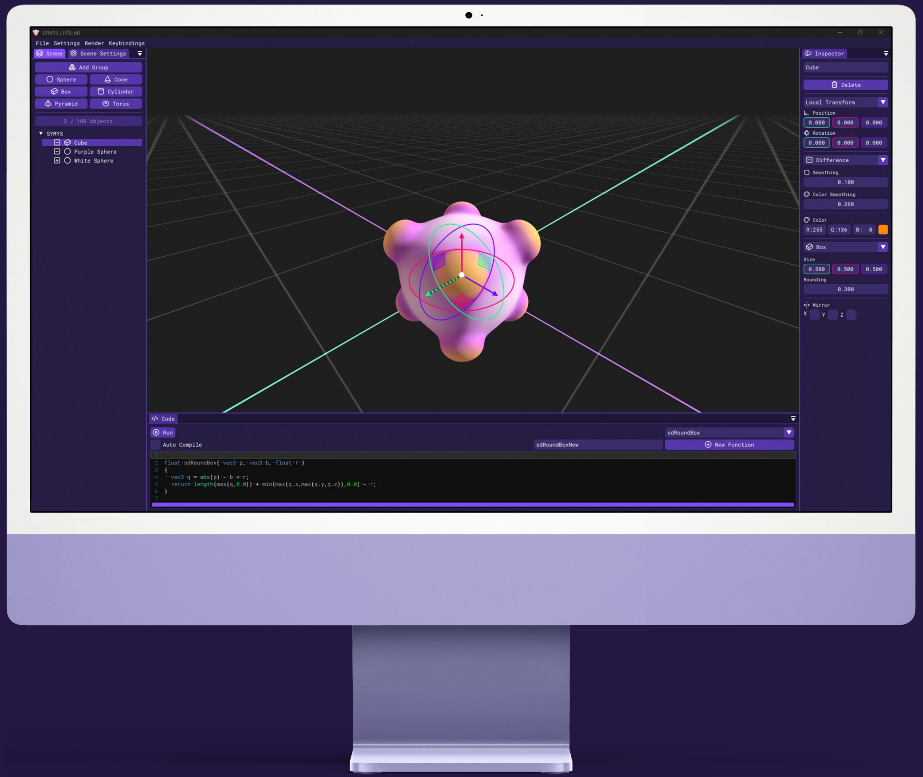
SYMYS



Cube
Local Transform
Position: 0.000 0.000 0.000
Rotation: 0.000 0.000 0.000
Difference
Smoothing: 0.100
Color Smoothing: 0.260
Color: R:255 G:136 B: 0
Box
Size: 0.500 0.500 0.500
Rounding: 0.300
Mirror: X Y Z

sdRoundBox
sdRoundBoxNew

```glsl

float sdRoundBox( vec3 p, vec3 b, float r )
{
  vec3 q = abs(p) - b + r;
  return length(max(q,0.0)) + min(max(q.x,max(q.y,q.z)),0.0) - r;
}

```

SYMYS - Mattia Metry

Overview

Symys is a C++ and Vulkan-based application designed to advance the field of 3D modeling by utilizing Signed Distance Fields (SDFs – detailed explanation on page 4) rather than traditional meshes and polygons. This innovative approach allows users to create various geometric primitives and combine them through Boolean operations such as Union, Difference, and Intersection. Additionally, Symys offers the capability to smooth these operations, facilitating the creation of organic models and structures.

To visualize SDFs on the screen, Symys employs a rendering technique called Sphere-Tracing (detailed explanation on page 4) . Users can render images and save them, enhancing the practicality of the application for ongoing projects and final outputs.

Symys provides a versatile environment with an intuitive GUI for ease of use, making it accessible to novice users. For more advanced users and developers, it offers the option to engage directly with the SDF definitions in shader code (GLSL) within the editor. This dual approach balances accessibility with detailed control, catering to a wide range of user expertise.

Additionally, Symys allows users to save scenes, ensuring they can continue their work seamlessly without having to start over each time. This functionality, combined with its advanced rendering capabilities, makes Symys a robust tool for both practical and experimental applications.

By bridging the gap between technical and artistic expertise, Symys represents a significant advancement in 3D modeling technology, providing a comprehensive platform for creating stunning models and scenes.

Context

SDFs describe surfaces of shapes through mathematical functions, allowing for manipulation using basic mathematical operations such as combining and adding functions to create more complex forms. While SDFs can be a powerful tool, they often require a deep mathematical understanding, which can be a barrier for many users.
Symys aims to bridge this gap between technical and artistic expertise. By offering extensive possibilities to work and interact with SDFs purely through a GUI, it enables 3D artists without mathematical knowledge to create stunning models and scenes. For more technical users and programmers, Symys provides the option to directly manipulate the GLSL code that defines the SDFs, granting them complete control over the shapes and forms.
Recent trends in the industry, such as Adobe's release of the beta version of Project Neo, the growing popularity of Womp3D, and the public alpha of ConjureSDF, a Blender plugin, indicate that SDFs are just beginning to gain traction. What sets Symys apart is its nature as a standalone application built from the ground up with SDFs in mind, running directly on the user's local machine. This design ensures high performance and usability.
Moreover, the ability to directly manipulate the SDF code within Symys offers users a level of control unmatched by other tools. This combination of a user-friendly GUI and detailed code control makes Symys a versatile and powerful tool for both artistic and technical users in the field of 3D modeling.

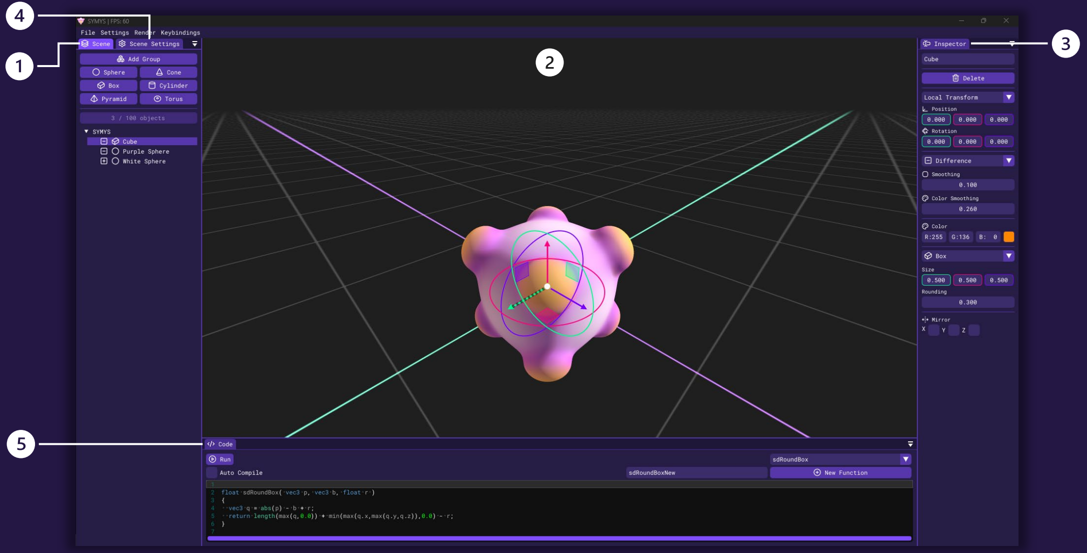
Example results

Made with SYMYS

User-friendly interface

A key feature of Symys is its user-friendly interface, designed to be intuitive for individuals with experience in 3D software. Symys uses a layout familiar to many other 3D tools, consisting of a ① ***Scene Hierarchy*** on the left side of the screen, a ② ***Viewport*** in the center, and an ③ ***Inspector*** on the right side. This arrangement ensures that users can quickly acclimate to the software.

Additionally, Symys includes a ④ ***Scene Settings*** tab on the left side, where users can adjust general settings such as background color, light direction, and camera position. The ⑤ ***Code*** Panel is located at the bottom of the screen, and it can be resized or fully hidden based on the user's preferences and needs.



Cube

Local Transform

Position: 0.000 0.000 0.000

Rotation: 0.000 0.000 0.000

Difference

Smoothing: 0.100

Color Smoothing: 0.260

Color: R:255 G:136 B: 0 🟧

Box

Size: 0.500 0.500 0.500

Rounding: 0.300

Mirror: X Y Z

sdRoundBox

sdRoundBoxNew

```
float sdRoundBox( vec3 p, vec3 b, float r )
{
  vec3 q = abs(p) - b + r;
  return length(max(q,0.0)) + min(max(q.x,max(q.y,q.z)),0.0) - r;
}
```

To interact with the 3D scene, Symys utilizes well-established 3D gizmos. The gizmo appears at the center of the selected element within the Viewport, allowing the user to move and rotate the element with ease.

To further enhance the user experience, Symys also supports a variety of shortcuts:

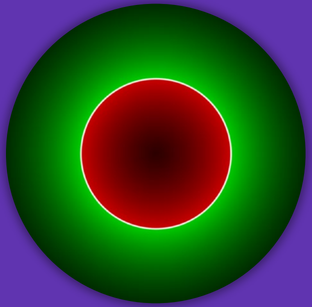
Key	Description
F	Focus on selected element
Delete	Delete selected element
Ctrl+C	Copy selected element
Ctrl+V	Paste selected element
Ctrl+D	Duplicate selected element
Ctrl+Z	Undo last action
Ctrl+Y	Redo last undone action
Ctrl+S	Save current scene
Ctrl+Shift+S	Save current scene as
Ctrl+N	Create new scene
Ctrl+O	Open existing scene
Shift+A	Open popup to add element

SDFs

Simply put, Signed-distance-fields are mathematical functions, which take in a point in space and return the distance to the nearest point on the surface.

The correct mathematical definition is:
SDFs are a special type of implicit surfaces. Implicit surfaces are a mathematical way to describe surfaces. In the case of Symys, we use 3 dimensional surfaces, which can generally be described by a function $f(p)\colon \mathbb{R}^3 \longrightarrow \mathbb{R}$ that takes in a point in 3D space p and returns a scalar value. By defining the function in such a way that the scalar value is zero at the surface, the function can be used to test if a point is on the surface, resulting in the following notation: $f(p) = 0$.
If we now add two additional constraints to the definition of the function, we get SDFs. The two new constraints are the following:

- $f(p) > 0$ if the point is outside the surface
- $f(p) < 0$ if the point is inside the surface



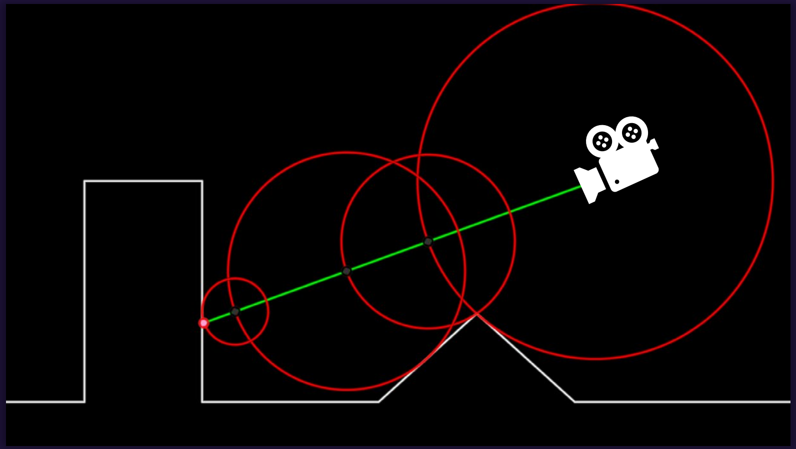
The image shows a 2D SDF of a circle.
Red means close to the surface inside,
Green means close to the surface outside,
and white is directly on the surface.

The function describing this circle is the following:

```
float sdCircle( in vec2 p, in float r )
{
    return length(p)-r;
}
```

Sphere-Tracing

To visualize SDFs on a screen, we use a rendering technique called ***Sphere-Tracing***. For each pixel on the screen, we define a position and a direction within the 3D scene, typically using the camera position and projection matrix. This position serves as the origin of a ray that we shoot into the scene along the specified direction. By calculating the distance to the nearest surface using the position and the combined SDF of all objects, we determine how far the ray can travel without hitting a surface. At this new position, we recalculate the distance and move the ray forward again. When the distance becomes small enough, indicating that the ray is close to a surface, we color the pixel accordingly.



2D representation of sphere-tracing. The green line represents the ray. The radius of the red circles represent the distance to the surface at each step.

Technical background

Sphere-tracing can be computationally intensive. To ensure optimal performance, the entire engine of Symys is built using C++, a language renowned for its performance and optimization capabilities. However, rendering complex SDF scenes in real time is beyond the capabilities of the CPU alone. To achieve real-time rendering, Symys leverages the GPU using the Vulkan API. This allows the use of compute shaders—programs that execute on the GPU in a massively parallel manner. This parallel processing enables the computation of thousands of pixels simultaneously, which are then combined into a single image and displayed as a frame.

In Symys, all actions occur within a defined scene, represented by a basic node structure known as a scene graph. A node can either be a group or an object. The root node is always a group, capable of having child nodes. Users can add new nodes within this root node to construct the entire scene. Whenever a major change occurs in the scene graph, the compute shader code is updated, recompiled, and pushed to the GPU.

Dynamic properties such as positions, scales, and colors are passed to the shader via uniform buffers. However, significant changes, such as adding or removing nodes, trigger the recompilation process. This ensures an optimal workload balance between the CPU and the GPU, maintaining high performance and responsiveness.

Symys was implemented using the following third-party libraries:

- ***ImGui*** : A bloat-free graphical user interface library. It provides most GUI elements needed, including buttons and input fields, which were essential for the UI of Symys.
- ***ImGuizmo*** : Integrates seamlessly with ImGui and provides 3D gizmos.
- ***ImGuiColorTextEdit*** : Extends ImGui by adding text editor functionality, which was used for the code-editing feature in Symys.
- ***Cereal*** : A library for serialization. It was used to serialize the scene to save it to disk and for the undo/redo system.
- ***LoadPNG*** : Use to export and save PNGs to the disk.
- ***TinyFileDialogs*** : Allows easy access to the operating system's file manager for saving and loading scenes.
- ***GLM*** : A math library that includes essential types and functions, such as vectors and matrices.
- ***Vulkan*** : Handles everything regarding the GPU, from executing shaders and pushing uniform buffers to synchronizing the GPU and CPU.
- ***SDL2*** : Used for low-level access to inputs such as keyboard and mouse events, as well as for creating and handling the window.

Reflection

Developing Symys has been an insightful journey into the realm of 3D modeling and rendering using Signed Distance Fields (SDFs). The project aimed to bridge the gap between technical and artistic expertise, providing a robust platform that caters to both novice users and experienced developers.
One significant challenge encountered was the computational intensity of sphere-tracing, a critical component for visualizing SDFs. To address this, I decided to use C++ and Vulkan for the first time. Learning such a low-level language and API provided new insights into how CPUs and GPUs work and how to efficiently manage their resources.
Another critical aspect was designing an intuitive and user-friendly interface. Symys features a layout familiar to those experienced with 3D software, including a Scene Hierarchy, Viewport, and Inspector. I initially underestimated the time needed to create the UI, but in the end, it was time well spent. Conversely, adding the code-editing feature turned out to be simpler than anticipated, which nicely balanced the development timeline.
In conclusion, the development of Symys has been a comprehensive learning experience, merging technical innovation with user-centered design. The application successfully bridges the gap between artistic creativity and technical precision, offering a powerful tool for 3D modeling and rendering. As SDF-based modeling continues to evolve, Symys is well-positioned to contribute significantly to this exciting field.

Future of SYMYS

The release of other SDF-based tools like Adobe's Project Neo, Womp3D, and ConjureSDF highlights the growing interest and potential of SDFs in 3D modeling. This trend has certainly motivated me to further develop Symys and add new features.

I already have an extensive list of potential new features and improvements that I'm eager to implement. These range from improving performance through the use of bounding volume hierarchy (BVH), expanding the library of primitives, and adding more advanced Boolean operations, to further enhancing the user interface.

Another exciting possibility is steering Symys towards becoming a game engine by adding in scripting support, enabling users to create interactive games and application directly within Symys.

Additionally, I am curious to explore how SDFs can be utilized and interacted with in virtual or mixed reality. Making SDFs more immersive could unlock significant untapped potential.

Regardless of the direction I choose to pursue, I am committed to continuing the development of Symys. This ongoing work will ensure that Symys remains a cutting-edge tool in the field of 3D modeling and rendering.